STRG-QL: Spatio-Temporal Region Graph Query Language for Video Databases

JeongKyu Lee; M. Emre Celebi

Jeongkyu Lee and M. Emre Celebi "STRG-QL: spatio-temporal region graph query language for video databases", Proc. SPIE 6820, Multimedia Content Access: Algorithms and Systems II, 68200P (January 28, 2008);

doi:10.1117/12.765531

http://dx.doi.org/10.1117/12.765531

STRG-QL: Spatio-Temporal Region Graph Query Language for Video Databases

Jeongkyu Lee[a], M. Emre Celebi[b]

[a]Department of Computer Science & Engineering
University of Bridgeport, Bridgeport, CT 06604-5692 U.S.A.

[b]Department of Computer Science
Louisiana State University in Shreveport, Shreveport, LA 71115 U.S.A.

ABSTRACT

In this paper, we present a new graph-based query language and its query processing for a Graph-based Video Database Management System (GVDBMS). Although extensive researches have proposed various query languages for video databases, most of them have the limitation in handling general-purpose video queries. Each method can handle specific data model, query type or application. In order to develop a general-purpose video query language, we first produce Spatio-Temporal Region Graph (STRG) for each video, which represents spatial and temporal information of video objects. An STRG data model is generated from the STRG by exploiting object-oriented model. Based on the STRG data model, we propose a new graph-based query language named STRG-QL, which supports various types of video query. To process the proposed STRG-QL, we introduce a rule-based query optimization that considers the characteristics of video data, i.e., the hierarchical correlations among video segments. The results of our extensive experimental study show that the proposed STRG-QL is promising in terms of accuracy and cost.

Keywords: Video database management systems, video query language, spatio-temporal region graph

1. INTRODUCTION

As a significant amount of video data has been generated in various areas, such as entertainment, business, education, science, and security, the demand for video database management system (VDBMS) to organize and manage it has increased recently. One of the main functions in this system is content-based video retrieval. Innumerable algorithms and techniques have been proposed to support this function, which can be classified into three categories: (1) visual feature based retrieval systems[1,2] which use visual features (i.e., color, shape, texture, etc.) of key frames to index frames, shots and scenes, (2) keyword based retrieval systems[3] which use manually extracted text information from video segments, and (3) object based retrieval systems which use spatio-temporal relationships among extracted objects.[4,5] A common challenge for the systems is to support the query processing for content-based query, search, and retrieval of video data.

Two main components of query processing are the query language and its processing. Many video query languages have been proposed, which can be divided into three groups as follows: (1) entirely new and specialized languages[6,7] which are developed to support domain specific applications and data models, (2) SQL-like query languages[8–11] which are based on some extensions of SQL, and (3) OQL-like query languages[12,13] which are based on some extensions of OQL. Table 1 shows the summary of various query languages mentioned above. As seen in the table, most works are designed for the particular data models and applications. In addition, only a limited number of query types are supported in each system.

In our previous work, we proposed a graph-based data structure, called *Spatio-Temporal Region Graph* (STRG) for video data.[14] An STRG represents spatial and temporal information among the objects extracted from the video sequence. To construct STRG from a video, a *Region Adjacency Graph* (RAG) is generated from each frame, and connected to each other temporally. An STRG provides a more flexible way to access video data, and supports various types of content-based queries.

(Send correspondence to J.L.) J.L.: E-mail: jelee@bridgeport.edu, M.E.C.: E-mail: ecelebi@lsus.edu

Multimedia Content Access: Algorithms and Systems II, edited by Theo Gevers, Ramesh C. Jain, Simone Santini, Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 6820 68200P, © 2008 SPIE-IS&T · 0277-786X/08/$18

Table 1. Summary of various query languages for video

Name	Basis	DBMS	Algebra	Target Data	Data Structure	Query Type	Visualizaton
VideoSQL [23]	SQL	OVID	X	Video Frame	Video Object	Content query	X
TVQL [13]	new	MMVIS	X	Video Event	X	Visual query	O
MOQL [19]	OQL	ObjectStore	Object Algebra	Video Object	X	Content query	X
GOQL [25]	OQL	Vstore	O-Algebra	Multimedia	Graph	Sequence query	X
CVQL [16]	new	VDBMS	X	Video Frame	Content Object	Content query	X
STQL [10]	SQL	MDBMS	Extend Relational	Moving Object	Extend Relational	Temporal query	O
VQP [1]	SQL	VDBMS	X	Stream Video	Streaming Data	Temporal join	X
Rule-based [26]	SQL	BilVideo	Extend Relational	Video Segment	X	Knowledge query	X

This paper proposes a new query language named STRG-QL and query processing. The motivation of STRG-QL and its processing is threefold. First, spatial and temporal information among video objects need to be represented properly. Second, the logical structure of video and hierarchical relationships between video segments (i.e., frame, object, shot, and video) need to be expressed appropriately. Third, a query language needs to be simple and flexible to handle various types of queries. To address these, we first define STRG data model using STRG and Object Definition Language (ODL).[15] An STRG data model can represent the logical structure of a video, and hierarchical relationships among video segments. Then, STRG-QL is developed to support various types of video queries. STRG-QL is an extension of Object Query Language (OQL)[15] by adding some functions and expressions. To process an STRG-QL, we introduce a rule-based query optimization which considers the characteristics of video data. Our contributions are summarized as follows:

- We introduce the formal definitions of graph-based video database managements including data model, query processing, and query optimization.
- We propose a new query language, STRG-QL that supports various query types such as Query by Example (QBE), and Query by Feature (QBF). In addition, we introduce several operational functions to handle STRG data model.
- We propose a rule-based query optimization considering the logical structure of video and hierarchical relationships among video segments, which provides more efficient STRG-QL processing.

The rest of the paper is organized as follows. In Section 2, we present Spatio-Temporal Region Graph (STRG) for video objects. An STRG data model is discussed in Section 3. The syntax and additional functions of STRG-QL are introduced with some example queries in Section 4. In Section 5, we provide a detailed discussion on the proposed rule-based STRG query processing strategy. The performance study is reported in Section 6. Finally, Section 7 presents some concluding remarks and future work.

2. SPATIO-TEMPORAL REGION GRAPH FOR VIDEO

In this section, we describe a graph-based video data structure, called *Spatio-Temporal Region Graph* (STRG) proposed in our previous work.[14] Then, we extend it to general hierarchical levels of video by adding STRG parsing system.

2.1 STRG Producing

For a given video, each frame is segmented into a number of regions using a region segmentation technique (EDISON).[16] Then, Region Adjacency Graph (RAG) is obtained by converting each region into a node, and spatial relationships among regions into edges. An RAG is good for representing spatial relationships among the nodes indicating the segmented regions. However, it cannot represent temporal characteristics of a video. We proposed a graph-based data structure for video objects, *Spatio-Temporal Region Graph* (STRG) in our previous work.[14] The STRG can handle both temporal and spatial characteristics of video object, which is defined as follows:

DEFINITION 1. *Given a video S, a Spatio-Temporal Region Graph, $Gst(S)$, is a six-tuple graph, $Gst(S) = \{V, E_S, E_T, \nu, \xi, \tau\}$, where V is a finite set of nodes for segmented regions from S, $E_S \subseteq V \times V$ is a finite set of spatial edges of S, $E_T \subseteq V \times V$ is a finite set of temporal edges of S, $\nu : V \rightarrow A_V$ is a set of functions generating node attributes, $\xi : E_S \rightarrow A_{E_S}$ is a set of functions generating spatial edge attributes, and $\tau : E_T \rightarrow A_{E_T}$ is a set of functions generating temporal edge attributes,*

In STRG, the node attributes (A_V) represent size (i.e., number of pixels), dominant color, and location of corresponding region, the spatial edge attributes (A_{E_S}) represent the relationships between two adjacent nodes such as spatial distance and orientation, and the temporal edge attributes (A_{E_T}) represent the relationships between corresponding nodes in two consecutive frames such as velocity and moving direction. Figure 1 (a) and (b) are actual frames in a sample video and their region segmentation results, respectively. Figure 1 (c) shows a part of STRG for frames #141 − #143 constructed by adding temporal edges which are horizontal lines between the frames.

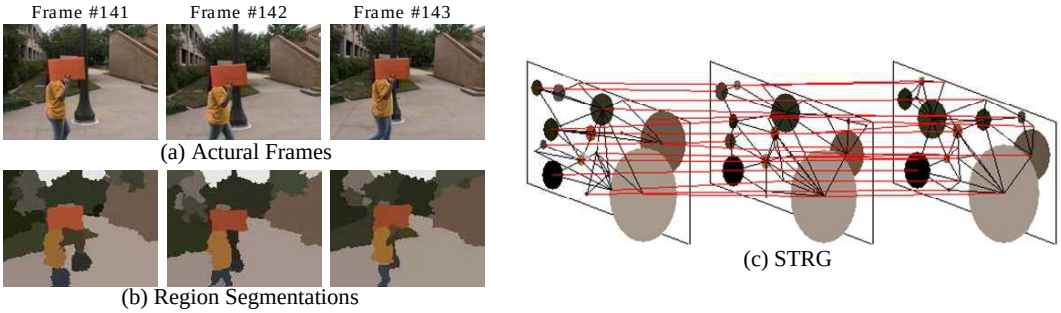


Figure 1. Example of STRG for frame #141 − #143

An STRG is an extension of RAGs by adding temporal edges (E_T) to them. E_T represents temporal relationships between two corresponding nodes in consecutive RAGs. Therefore, the main procedure of building STRG is how to construct E_T. To find the corresponding nodes in two consecutive RAGs, we use a graph-based tracking algorithm developed in our previous work.[14]

2.2 STRG Parsing

After we generate an STRG for a video, we need to partition it into shots for higher level content analysis and indexing. Although an STRG provides important information about a video such as spatial and temporal relationships among objects, it still requires high computational complexity due to various graph operations. A video shot, defined as a collection of interrelated consecutive frames taken contiguously by a single camera operation, is meaningful to serve as an elementary unit to address this. Shot boundaries can be detected by employing a metric to measure the difference between two consecutive frames. This is based on the fact that the content remains nearly the same in a shot. If the difference is less than a certain threshold value, two frames are considered to be in the same shot. To measure the difference, we compute a temporal connectivity between two consecutive RAGs (frames), which is the number of temporal edges (E_T) between two consecutive RAGs. If the temporal connectivity exceeds more than a certain threshold value, the two frames corresponding to the two RAGs are considered to be in the same shot. Otherwise, the two frames are considered to be in two different shots. After detecting shot boundaries of an STRG, we compute a key RAG (key frame) of each shot to characterize it.

Each segmented STRG (i.e., shot) is decomposed into *Object Graphs* (OGs) and a *Background Graph* (BG). An OG represents an object moving over frames, which can be defined as a set of sequential nodes connected to each other by a set of temporal edges (E_T). An OG is a subgraph of STRG such that the spatial edge set E_S is empty. However, due to the limitations of region segmentation techniques, different color regions belonging to a single object cannot be always detected as a single region. For instance, a body of a person may consist of several regions such as the head, the upper body and the lower body. In order to merge two OGs which belong to a single object, we consider the attributes (i.e. velocity and moving direction) of the temporal edges (E_T). If two OGs have same moving direction and the same velocity, these can be merged into one.

A video frame usually consists of two areas: foreground and background. A foreground is the main target on which a camera focuses, and a background is a supporting area that does not change significantly over the time in a shot. Generally speaking, it is sufficient to maintain only one *Background Graph* (BG) for each segmented STRG (shot) where there is little difference in the background over the frames. The main idea of building a BG is inspired by adaptive background modeling and mosaic techniques.[17] In order to obtain a background graph for each shot, we first subtract all OGs from STRG, and overlap all subtracted RAGs which belong to the same shot by matching temporal edges (E_T).

3. STRG DATA MODEL

In this section, we define *STRG data model* based on the object-oriented data model. In order to facilitates high level manipulations of video content, we propose STRG data model using an ODMG ODL schema.[15] First, we define four basic classes of STRG data model as seen in Figure 2; **Node**, **Edge**, **RAG**, and **STRG**. Each class represents both its class name and a type of objects belonging to the class.

```
class Node (extent Nodes)
{ attribute Short location;
  attribute Short size;
  attribute Short color;
};

class Edge (extent Edges)
{ attribute String e_type;
  attribute set<Node> nodes;
  attribute Double velocity;
  attribute Double direction;
};

class RAG (extent RAGs)
{ attribute set<Node> nodes;
  attribute set<Edge> Sedges;
};

class STRG (extent STRGs
{ attribute set<Node> nodes;
  attribute set<Edge> sedges;
  attribute set<Edge> tedges;
};
```

Figure 2. Four basic classes in STRG data model

- **Node** type is an ordinary object type which represents a segmented region in a video frame. It can have the attributes in which each value is a basic type, i.e., short. Objects of **Node** types are the nodes of **RAG** and **STRG**. A **Node** represents a single semantic object in a frame.
- **Edge** type is a sequence type, and its base type is a **Node** type. Objects of **Edge** type must be sequences of two **Node**s. It can have the additional attributes in which each value is a basic type such as string and double. Objects of **Edge** types are the edges of **RAG** and **STRG**. An **Edge** represent either spatial relationship (adjacency) or temporal relationship (tracking) between two objects in a video.
- **RAG** type is a tuple type with two special attributes, **Node** and **Edge** which contain a set of node and edge objects, respectively. A **RAG** represents spatial information among objects in a frame.
- **STRG** type is a tuple type with three special attributes, **Node** and two types of **Edge**s. One **Edge** is for spatial edges, and the other is for temporal edges. In an STRG object, its **Node** and **Edge** attributes contain a set of node and edge objects, respectively. A **STRG** represents spatial and temporal relationships among objects in a video segment.

We now define the ODL schema which describes a video database including videos, shots, moving objects, and backgrounds. Figure 3 shows four classes of the schema: **Video**, **Shot**, **OG** and **BG**. For every class we declare an extent, which refers to the current collection of all objects in that class. Each raw video is modeled as **Video** object. A **Video** provides some attributes describing the basic characteristics of a raw video data, i.e., video name and a set of shots that a video contains. A **Shot** models each segmented shot in a video. It provides some attributes describing a set of RAGs representing frames, a key RAG (frame), and moving objects and background belonging to it. An **OG** and a **BG** model individual object graph and background graph respectively, mentioned in Section 2.

4. STRG QUERY LANGUAGE (STRG-QL)

In this section, we propose *STRG Query Language* (STRG-QL) from STRG data model. STRG-QL is an OQL-like query language to design, manipulate and query video objects. The STRG-QL is a superset of OQL. It recognizes the syntax of OQL but it also provides additional operators, functions and predicates to manipulate STRG data model.

```
class Video (extent Videos
             key v_id)
{ attribute Short v_id;
  attribute String name;
  relationship set<Shot> shots
       inverse Shot::video;
  other attributes;
};
```

```
class Shot (extent Shots
            key s_id)
{ attribute Short s_id;
  attribute set<RAG> frames;
  attribute RAG keyframe;
  relationship Video video
       inverse Video::shots;
  relationship set<OG> ogs
       inverse OG::shot;
  relationship BG bg
       inverse BG::shot;
  other attributes;
};
```

```
class OG (extent OGs key o_id)
{ attribute Short o_id;
  attribute set<Node> nodes;
  attribute set<Edge> tedges;
  relationship Shot shot
       inverse Shot::ogs;
  other attributes;
};
```

```
class BG (extent BGs key b_id)
{ attribute Short b_id;
  attribute set<Node> nodes;
  attribute set<Edge> sedges;
  attribute set<Edge> tedges;
  relationship Shot shot
       inverse Shot::bg;
  other attributes;
};
```

Figure 3. ODL schema of GVDBMS using STRG model

As in OQL, STRG-QL uses the traditional *"select ... from ... where ..."* statements for querying. However, each clause has an extended meaning in terms of graphs. An STRG-QL query can be expressed by the following structure.

- SELECT: target of query
- FROM: range (or source) of query
- WHERE: condition (or predicate) of query

In SELECT clause, the target of a query is specified, i.e., videos (**Videos**), shots (**Shots**), moving objects (**OGs**), backgrounds (**BGs**), etc. Also, video browsing functions can be used in SELECT clause, such as SUMMARY(). The range (or source) of query is specified in FROM clause, which defines the search space of a query. In addition, a query video given by a user is specified in this clause, which makes STRG-QL support Query by Example. The qualification of a query is specified in WHERE clause. Objects in the FROM clause are evaluated by the specified predicates to get the results. Here is a typical example of STRG-QL.

Q1: Find key frames of all shots which belong to the video id = 1

SELECT s.keyframe **FROM** s **in** Shots **WHERE** s.vid = 1;

This simple query retrieves the key frame of each shot. Full descriptions of STRG-QL syntax will be discussed in the following subsection.

4.1 Extended Functions in STRG-QL

The proposed extended functions in STRG-QL are related to graph operations (i.e., graph matching and subgraph isomorphism) and video presentations (i.e., video summary). Unlike existing video query languages which considered spatial or temporal predicates, the proposed STRG-QL does not need to consider those predicates since STRG data model implicitly includes spatial and temporal information. In the following, new functions in STRG-QL are introduced.

4.1.1 GDM()

In order to compute the distance (dissimilarity) between two RAGs or BGs, we define a graph matching algorithm, called *Graph Dissimilarity Measure* (GDM), which uses the *maximal common subgraph*.[18] The graph dissimilarity measure GDM between two graphs G_1 and G_2 can be defined as follows.

DEFINITION 2. *The Graph Dissimilarity Measure (GDM) between G_1 and G_2 is defined as:*

$$GDM(G_1, G_2) = 1 - \frac{|G_C|}{max(|G_1|, |G_2|)}$$

where $|G|$ denotes the number of nodes of G, and G_C is the maximal common subgraph of G_1 and G_2. In Definition 3, G_C can be computed based on maximal clique detection.[19] In GDM(), the possible operands are any types of graphs (i.e., BG, RAG or STRG).

This query return all shots whose background and key frame are the same. Using GDM() to compare two RAGs or BGs, we allow a certain error margin to find similar graphs, since it is rare to find identical graphs in STRG data model. In this case, we use a certain threshold value (δ) for the error margin.

Q2: Find s_id of all shots of which background and key frame are same.

SELECT b.shot.s_id **FROM** b **in** BGs **WHERE** GDM(b.nodes, b.shot.keyframe.node) < δ;

4.1.2 GED()

Since GDM() is designed for the general purpose of graph matching such as RAGs and BGs, it is not suitable to use GDM() for OGs which are time-varying data. In order to address this, we consider the temporal characteristic of OG to compute the distance (dissimilarity) between two OGs. Since the edit distance can handle not only graph matching, but also time-varying data, we extend it to *Graph Edit Distance* (GED). Since the main operations to edit graphs deal with nodes and their attributes rather than edges, we consider only the nodes and their attributes. Let $OG^s_m = \{v^s_1, \ldots, v^s_m, \nu^s\}$ and $OG^t_n = \{v^t_1, \ldots, v^t_n, \nu^t\}$ be s^{th} and t^{th} OGs with m and n number of nodes, respectively. The distance function GED between OG^s_m and OG^t_n can be defined as follows.

DEFINITION 3. *The Graph Edit Distance (GED) between OG^s_m and OG^t_n is defined as:*

$$GED(OG^s_m, OG^t_n) = \begin{cases} \sum_{i=1}^{m} |v^s_i - g_i| & \text{if } n = 0, \\ \sum_{i=1}^{n} |v^t_i - g_i| & \text{if } m = 0, \\ min[GED(OG^s_{m-1}, OG^t_{n-1}) + dist(v^s_m, v^t_n), & \\ \quad GED(OG^s_{m-1}, OG^t_n) + dist(v^s_m, gap), & \\ \quad GED(OG^s_m, OG^t_{n-1}) + dist(gap, v^t_n)] & \\ \quad otherwise. \end{cases}$$

where gap is an added, deleted or changed node, and $g_i = \frac{v_{i-1}+v_i}{2}$ is a gap for i^{th} node. And,

$$dist(v^s_i, v^t_j) = \begin{cases} |v^s_i - v^t_j| & \text{if } v^s_i, v^t_j \text{ are not a gap} \\ |v^s_i - g_j| & \text{if } v^t_j \text{ is a gap} \\ |v^t_j - g_i| & \text{if } v^s_i \text{ is a gap.} \end{cases}$$

For better readability, let v indicate a value $\nu(v)$ of node attribute. *dist* is the cost function for editing nodes.

Q3: Find o_id of all moving objects which have the same moving pattern to moving objects in the shot id = 1

SELECT o.o_id **FROM** o **in** OGs, q **in** (**select** a.ogs **from** a **in** Shots **where** a.s_id = 1)
WHERE GED(o.nodes, q.nodes) < ε;

Q3 gives a list of OGs whose moving pattern is same as those of the shot (s_id = 1) by GED(). We allow a certain threshold value (ε) for the error margin like using δ of GDM().

4.1.3 SUMMARY()

Since consecutive frames in a segmented shot are little different, it is efficient to summarize a long video without loss of main semantics when the frames are compared or browsed. In order to summarize videos or shots, we exploit a scene tree construction technique proposed in,[20] and extend it to STRG data model. SUMMARY() requires three parameters as input, i.e., source object (a set of frames, shots or videos), summary level (L), and summary length (T_{len}). It returns a set of frames which is a summary without loss of original semantics. The detailed algorithm for the summarization can be seen in.[20]

Q4: Summarize a video id = 1 with Level = 1 and T_{len} = 0.9

SELECT SUMMARY(v.shots, 1, 0.9) **FROM** v **in** Video **WHERE** v.v_id = 1;

Q4 returns the summary of a video. Moreover, it can be used in FROM and WHERE clauses to reduce the processing time.

4.2 Supported Query Types

In this section, we present two main query types that STRG-QL supports; Query by Feature (QBF), and Query by Example (QBE). While QBF is the basic query type which is compatible to ODMG OQL, QBE supports a query with a sample video clip. We provide some examples of each query type.

4.2.1 Query By Feature

This type of query is used to retrieve salient objects from database that satisfy the conditions given by feature values. Since the query uses feature values of objects in database, it is fully compatible to a standard OQL. A typical example of QBF is given below:

Q5: Find moving objects of all OGs of which node has the following trajectory: ([10,60], [30,60], [50,60], [70,60], [90,60], [110,60])

```
SELECT o.o_id FROM o in OGs  WHERE GED(o.nodes, set(struct Node{ [10,60], nil, nil },
struct Node{ [30,60], nil, nil }, ……., struct Node{ [110,60], nill, nil })) < ε;
```

In this query, a set of Nodes is constructed from given feature values by following the locations of the moving object. The constructed nodes are compared to each OGs.nodes to find salient objects. However, it is inconvenient to make a query statement using feature values. Sometimes, it is not possible to make an appropriate query statement from given conditions. In order to address this, we propose more convenient query type using examples.

4.2.2 Query By Example

Query By Example (QBE) was developed by IBM in the 1970s to help users in their retrieval of information from the database using query templets.[21] We employ the idea of QBE to address the limitation of QBF. QBE makes users retrieve data using a sample video clip. In other words, a sample video clip instead of feature values is given to a query. To support a query video, the FROM expression in OQL grammar is extended as follows.

```
<from_clause> := <variable_name> IN <expression>
        | <variable_name> IN <expression>, <from_clause>
        | <variable_name> IN <expression> OF <videolist>
<videolist> := [<videolist> ‘,’] <name>
<name> :=  '[A-Za-z][A-Za-z0-9_#]*'
```

The query videos are assigned to OF clause. When a query has multiple videos, a comma is used as the separator. If OF clause is recognized in a compile time, a query processor creates temporary objects for a query video (i.e., **tmpVideos**, **tmpShots**, **tmpOGs** and **tmpBGs**) using STRG data model in memory. During the query execution phase, each data object is extracted from the query videos before retrieval. The temporary objects are removed from the memory after the query is completed. The temporary objects, **tmpVideos**, **tmpShots**, **tmpOGs** and **tmpBGs**, are inherited from **Videos**, **Shots**, **OGs** and **BGs**, respectively.

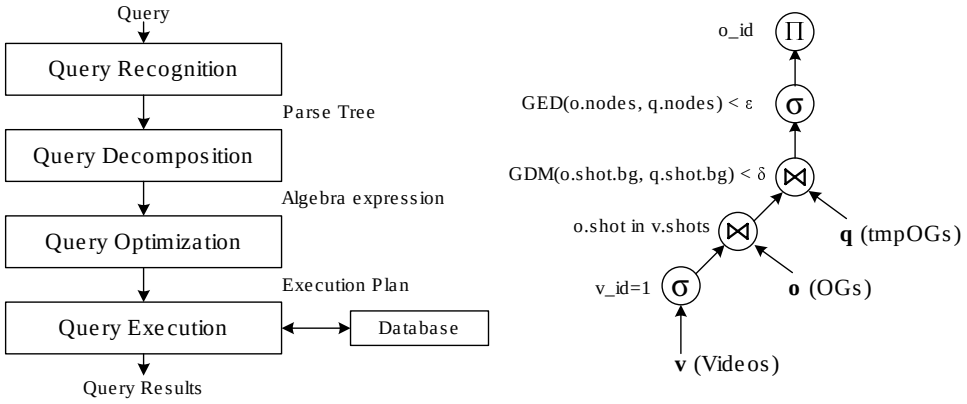
Q6: Find moving objects of OGs in a video with v_id = 1 whose moving patterns and background are the same as moving objects and background in a given video clip (query.avi)

```
SELECT  o.o_id FROM  v in Videos,  o in OGs, q in tmpOGs of ‘query.avi’
WHERE  v_id = 1 and o.shot in v.shots and GED(o.nodes, q.nodes) < ε
    and  GDM(o.shot.bg, q.shot.bg) < δ;
```

In Q6, all moving objects in a query video (i.e., ‘query.avi’) are first extracted by STRG producing module, and will be stored at **tmpOG** as OG an type. Then, the remaining query is processed like a normal query. Other STRG objects, such as **tmpVideo**, **tmpShot**, and **tmpBG**, can be created in memory if needed.

5. STRG QUERY PROCESSING

An STRG-QL query is processed in the query processor to retrieve data from a database. Figure 4 (a) shows four main phases of STRG query processing: query recognition, query decomposition, query optimization, and query execution. Query recognition performs syntactic and semantic checks of input query, and generates a parse tree. Query decomposition constructs a query tree from a parse tree using algebra expressions. The query tree is optimized for an efficient execution in query optimization phase. Finally, the query execution retrieves the data in database according to the execution plan. A set of results is returned to the user who requested the query. Among query processing phases, the most important one is the query optimization since it determines the overall performance of STRG query processing. In this section, we focus on the query optimization strategy, i.e., a rule-based optimization strategy of STRG-QL is discussed.



(a) STRG Query processing phases (b) Execution plan of Q7

Figure 4. STRG Query processing phases and an example of execution plan

A query optimization is the activity of choosing an efficient execution strategy for processing a given query. Since STRG data model deals with video data which have the hierarchical characteristics, such as video, shot, and frame, we need a different strategy to process STRG-QL. For example, suppose that a query includes two join operations such as video to video or frame to frame levels. A conventional query processing uses statistical information to reduce the cost of these operations. However, video level operations can be processed in advance without investigating additional information because videos always have lower cardinality than frames do. Therefore, using logical and hierarchical structure of a video not only saves the cost of a query optimization, but improves performance of a query execution. The rules of query optimization in STRG-QL are as follows:

Rule 1: Perform the selection operation as early as possible. Like other query optimization, selection can reduce the cardinality of object. Therefore, it makes sense to move the selection operations as far down the tree as possible.

Rule 2: Perform the operations related to OGs after all operations related to Video and Shot operation. Since OG is an object with high cardinality, OG should be cascaded before it is processed. In order to do this, Rule 2 moves the operations related to OG later than others.

Rule 3: Perform extended operations of STRG-QL as late as possible. In general, the extended functions of STRG-QL (i.e., GDM(), GED(), and SUMMARY()) mentioned in Section 4.1 have a significant amount of processing compared with conventional operations of OQL. Therefore, it is better to perform them as late as possible to reduce the cost of extended functions.

Rule 4: Use STRG-Index for the operations related to Shot, BG and OG. In order to avoid a sequential scanning in selection, and cartesian product in join operation, we use an STRG-Index mentioned in previous subsection.

Using the above rules, we build an execution plan for the example query Q6 in Section 4. First, Q6 query is decomposed into the following subqueries:

- Subquery 1: v_id = 1 and o.shot in v.shots
- Subquery 2: GED(o.nodes, q.nodes) $< \varepsilon$
- Subquery 3: GDM(o.shots.bg, q.shots.bg) $< \delta$

By Rules 1 and 2, Subquery 1 is executed first. Then, Subquery 3 performs a join operation with GDM() between the output of Subquery 1 and **q** (tmpOGs) according to Rule 3. Subquery 2 is carried out after Subquery 3 because of Rule 2. Figure 4 (b) shows the execution plan constructed by our rule-based query optimization. For the selection (σ) and the join operations ($\bowtie$), two STRG-Index for **v** and **q** can be used for more efficient query processing.

6. EXPERIMENTAL RESULTS

In our experiments, we conduct some evaluations of the proposed STRG-QL and its processing as follows:

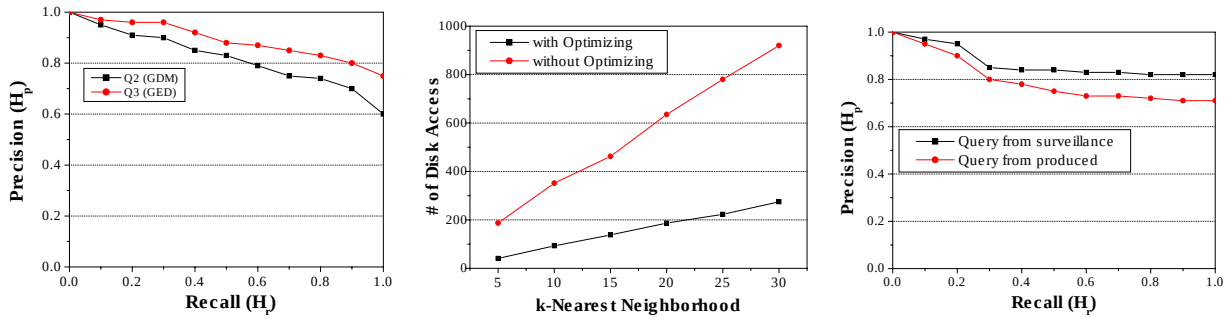
- The performance of proposed functions (GDM() and GED()).
- The performance of a rule-based optimization strategy.
- The overall retrieval accuracy for STRG-QL.

To assess these, we employ recall/precision, computation time, and disk access. We compare queries with and without the STRG-Index using the computation time. Also, we compare queries with and without the rule-based optimization using the number of disk accesses.

We implement a middleware which supports STRG-QL. If a query processor recognizes any STRG-QL specific expression during a query recognition phase, a given query is processed in the middleware. Otherwise, the query is processed as a normal OQL query. We use the same query examples mentioned throughout this paper, and modify some of them when needed. All experiments are performed on an Intel Pentium IV 2.6 GHz CPU using Java.

Table 2. Description of real video data set

Group	Type	Name	Duration (hh:mm:ss)	Number of Shots	Number of OGs	Complexity of Object Motion
Produced Video	Drama	Silk Stalking	00:10:24	96	151	Very Low
	Sitcom	Friends	00:10:22	117	289	Low
	Talk Show	Divorce Court	00:11:11	161	197	Very Low
	Si-Fi	Star Trek	00:12:27	112	298	Low
Surveillance Video	Indoor	Laboratory 1	40:38:02	1	411	High
	Indoor	Laboratory 2	04:12:24	1	147	High
	Outdoor	Traffic 1	00:15:08	1	195	Very High
	Outdoor	Traffic 2	00:12:48	1	203	Very High
		Total	46:02:46	490	1891	



(a) GDM() and GED() (b) Optimization (c) Accuracy

Figure 5. Performance of STRG-QL: (a) GDM() and GED(), (b) STRG-QL Optimization, and (c) Accuracy

6.1 Data Sets

We use both synthetic data and real video streams in our experiments.

Synthetic data set: To demonstrate the performance of the proposed functions and STRG-Index in various ways, the synthetic data is generated and used for the experiments. The synthetic data include only OGs because it is not practical to synthesize a whole STRG. Since an OG is a type of time-series data, we generate new data by combining the Pelleg data set[22] which is widely used to test clustering algorithms, with the Vlachos data set[23] which is 2-D time-series data with noise. Our synthetic data is generated as follows. First, we design 48 moving patterns: vertical (12), horizontal (12), diagonal (8) and U-turn (16). Each pattern has two directions, different size of objects and various time lengths. Second, we generate time-series data by following the approach described in.[22] Here, the data set has 48 moving patterns and is distributed by Gaussian with $\sigma = 5$. Third, we add some noise to each data point based on the work in.[23] Finally, the generated data is converted to object graphs using the nodes, temporal edges, and their attribute values.

Real video data set: To evaluate the effectiveness of the proposed approach, the real data set consists of two different groups: the first group has the produced videos, and the other has the surveillance videos. Table 2 shows the description of the real video data used in the experiment. For the produced videos, we choose four video clips from,[20] which have various types of content and shot changes. The surveillance videos are taken from the inside of a laboratory (Laboratory 1, 2) and outside for traffic scenes (Traffic 1, 2). As seen in Table 2, while the produced videos have various types of shot changes and simple object motions, the surveillance videos have complex patterns of object motions without shot change. Our video data set is about 46 hours, which is long enough to evaluate the proposed algorithms.

6.2 Performance of STRG-QL

We first evaluate the performance of the proposed graph operational functions (GDM() and GED()) on the synthetic data set, which are the main features to decide the performance of STRG-QL. Example query Q2 and Q3 are used for the evaluation of GDM() and GED(), respectively, since Q2 and Q3 have those functions. Figure 5 (a) shows the precision-recall values for two queries. The precision of Q2 (GDM()) is over 60% over any values of recall. And, the precision of Q3 (GED()) is over 80% over any values of recall. These indicate that the proposed functions (GDM() and GED()) are the optimal in finding similar objects. The accuracy of GDM() is a little less than that of GED() since GDM() is designed for general purpose.

Figure 5 (b) shows the number of disk accesses to validate the efficiency of the rule-based query optimization. We use the same k-NN query in the previous subsection. We count the number of disk accesses when a query is executed with and without the query optimization. As seen in the figure, the rule-based query optimization can reduce the significant number of disk accesses, which makes the performance of STRG-QL better.

To demonstrate the overall accuracy of STRG-QL and its query processing, we use the real videos in Table 2. First, all the videos are parsed as STRG, and the outputs are stored at the repository as the forms of STRG data model, i.e., Video, Shot, OG and BG. The fifth and sixth columns of Table 2 indicate the number of detected

SELECT o.o_id **FROM** o **in** OGs, q **in** tmpOGs **of** 'query.avi'
WHERE GED(o.nodes, q.nodes) < ε **and** GDM(o.shot.bg, q.shot.bg) < δ **and rownum** < k
ORDER BY GED(o.nodes, q.nodes) **asce;**

Shots and OGs, respectively. We use a k-NN query which is an extension of the example query Q6. The k-NN query can be expressed by STRG-QL where k is the returned data, $\varepsilon = 0.1$ and $\delta = 0.1$ as follows.

Two query videos are selected from the database; the first query video is from a surveillance group, and the second is from a produced group in Table 2. The results are verified with the ground truth. As seen in Figure 5 (c), we observe that the precision of the first query is over 80%, and that of the second query is over 70% at any values of recall. In other words, the proposed STRG-QL provides consistent accuracy for various video types.

Figure 6 shows results of the first query when $k = 2$. The first row shows some selected frames of the query video. Two matched videos are in the second and third rows. As seen in Figure 6, STRG-QL is able to find the objects with similar moving patterns.

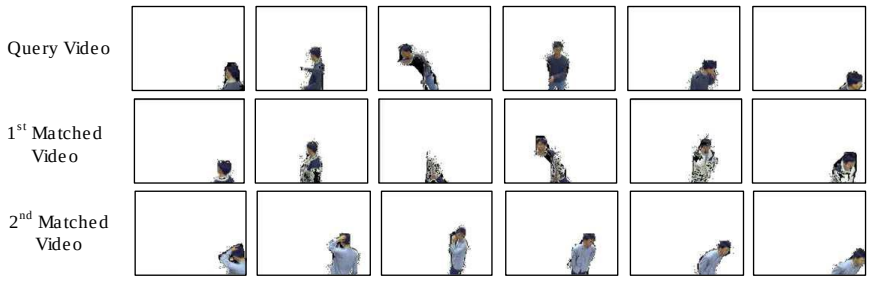


Figure 6. Result of k-NN query using STRG-QL (k=2)

7. CONCLUDING REMARKS

In this paper, we introduce a graph-based query language (STRG-QL), and its query processing for content-based video retrieval system. In order to develop a general-purpose video query language, first we model video data using Spatio-Temporal Region Graph (STRG). Based on the STRG data model, we propose a new graph-based query language named STRG-QL, which supports various types of video queries. To process the STRG-QL, we introduce a rule-based query optimization which considers the characteristics of video data. Some query examples are provided to illustrate STRG-QL and its query processing. We have developed a system, called Graph-based Video Database Management System (GVDBMS) using the proposed approaches. Experimental results on both synthetic data and real video streams show the proposed approaches are promising.

Future work is directed towards improving the graph operational functions in terms of accuracy. We are also planning to exploit Ontology techniques[24] to support a query by concept.

REFERENCES

1. A. Hanjalic, R. L. Lagendijk, and J. Biemond, "Automated high-level movie segmentation for advanced video-retrieval systems," *IEEE Trans. on Circuits and Systems for Video Technology* **9**(4), pp. 580–588, 1999.
2. W. Mahdi, M. Ardebilian, and L. M. Chen, "Automatic video scene segmentation based on spatial-temporal clues and rhythm," in *International Journal of Networking and Information Systems*, January 2001.
3. R. Hjelsvold and R. Midtstraum, "Modelling and Querying Video Data," in *Proceedings of the 20th International Conference on Very Large Databases*, pp. 686–694, (Santiago, Chile), 1994.
4. S.-C. Chen and R. L. Kashyap, "A Spatio-Temporal Semantic Model for Multimedia Database Systems and Multimedia Information Systems," *IEEE Transactions on Knowledge and Data Engineering* **13**(4), pp. 607–622, 2001.

5. L. Chen and M. T. Özsu, "Multi-Scale Histograms for Answering Queries over Time Series Data," in *Proceedings of the 20th International Conference on Data Engineering*, p. 838, (Boston, MA), 2004.
6. S. Hibino and E. A. Rundensteiner, "MMVIS: Design and Implementation of a Multimedia Visual Information Seeking Environment," in *ACM Multimedia*, pp. 75–86, 1996.
7. T. C. T. Kuo and A. L. P. Chen, "Content-Based Query Processing for Video Databases," *IEEE Transactions on Multimedia* **2**(1), pp. 1–13, 2000.
8. E. Oomoto and K. Tanaka, "OVID: Design and Implementation of a Video-Object Database System," *IEEE Trans. Knowl. Data Eng.* **5**(4), pp. 629–643, 1993.
9. M. Erwig and M. Schneider, "Spatio-Temporal Predicates," *IEEE Trans. Knowl. Data Eng.* **14**(4), pp. 881–901, 2002.
10. W. G. Aref, M. A. Hammad, A. C. Catlin, I. F. Ilyas, T. M. Ghanem, A. K. Elmagarmid, and M. S. Marzouk, "Video query processing in the VDBMS testbed for video database research," in *MMDB*, pp. 25–32, 2003.
11. Ö. Ulusoy, U. Güdükbay, M. E. Dönderler, E. Saykol, and C. Alper, "BilVideo Video Database Management System," in *VLDB*, pp. 1373–1376, 2004.
12. J. Z. Li, M. T. Özsu, and D. Szafron, "Modeling of Moving Objects in a Video Database," in *ICMCS*, pp. 336–343, 1997.
13. L. Sheng, Z. M. Özsoyoglu, and G. Özsoyoglu, "A Graph Query Language and Its Query Processing," in *ICDE*, pp. 572–581, 1999.
14. J. Lee, J. Oh, and S. Hwang, "STRG-Index: Spatio-Temporal Region Graph Indexing for Large Video Databases," in *Proc. of 2005 ACM SIGMOD Intl. Conf. on Management of Data*, pp. 718–729, (Baltimore, MD), June 2005.
15. R. G. G. Cattell and D. K. Barry, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann, 2000.
16. D. Comaniciu and P. Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(5), pp. 603–619, 2002.
17. R. C. Jones, D. Dementhon, and D. S. Doermann, "Building mosaics from video using mpeg motion vectors," *ACM Multimedia* , pp. 29–32, March 1999.
18. H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters 19* , pp. 255–259, 1998.
19. G. Levi, "A note on the derivation of maximal common subgraphs of two directed or undirected graphs," *Calcols 9* , pp. 341–354, 1972.
20. J. Oh and K. A. Hua, "An efficient and cost-effective technique for browsing and indexing large video databases," in *Proc. of 2000 ACM SIGMOD Intl. Conf. on Management of Data*, pp. 415–426, (Dallas, TX), May 2000.
21. M. M. Zloof, "Query-By-Example," *IBM Systems Journal* **16**(34), pp. 324–343, 1977.
22. D. Pelleg and A. Moore, "X-means: Extending K-means with Efficient Estimation of the Number of Clusters," in *Proc. of the 7th International Conference on Machine Learning*, pp. 727–734, (San Francisco), 2000.
23. M. Vlachos, D. Gunopulos, and G. Kollios, "Robust similarity measures for mobile object trajectories," in *Proc. of 5th International Workshop Mobility In Databases and Distributed Systems*, pp. 721–728, (Aix-en-Provence, France), September 2002.
24. N. Noy and M. D.L., "Ontology development 101: A guide to creating your first ontology," in *Proc. of 19th Int'l Conf. on Conceptual Modeling (ER2000)*, pp. 383–396, October 2000.